Fully Characterizing Lossy Catalytic Computation

Marten Folkertsma
CWI, Amsterdam, The Netherlands
QuSoft, Amsterdam, The Netherlands

Ian Mertz
University of Warwick, UK

Florian Speelman
University of Amsterdam, The Netherlands

Quinten Tupker
CWI, Amsterdam, The Netherlands

Abstract

A *catalytic machine* is a model of computation where a traditional space-bounded machine is augmented with an additional, significantly larger, "catalytic" tape, which, while being available as a work tape, has the caveat of being initialized with an arbitrary string, which must be preserved at the end of the computation. Despite this restriction, catalytic machines have been shown to have surprising additional power; a logspace machine with a polynomial length catalytic tape, known as *catalytic logspace* (CL), can compute problems which are believed to be impossible for L.

A fundamental question of the model is whether the catalytic condition, of leaving the catalytic tape in its exact original configuration, is robust to minor deviations. This study was initialized by Gupta et al. (2024), who defined *lossy catalytic logspace* ($\mathsf{LCL}[e]$) as a variant of CL where we allow up to e errors when resetting the catalytic tape. They showed that $\mathsf{LCL}[e] = \mathsf{CL}$ for any $e = O(1)$, which remains the frontier of our understanding.

In this work we completely characterize lossy catalytic space ($\mathsf{LCSPACE}[s, c, e]$) in terms of ordinary catalytic space ($\mathsf{CSPACE}[s, c]$). We show that

$$\mathsf{LCSPACE}[s, c, e] = \mathsf{CSPACE}[\Theta(s + e \log c), \Theta(c)]$$

In other words, allowing e errors on a catalytic tape of length c is equivalent, up to a constant stretch, to an equivalent errorless catalytic machine with an additional $e \log c$ bits of ordinary working memory.

As a consequence, we show that for any e, $\mathsf{LCL}[e] = \mathsf{CL}$ implies $\mathsf{SPACE}[e \log n] \subseteq \mathsf{ZPP}$, thus giving a barrier to any improvement beyond $\mathsf{LCL}[O(1)] = \mathsf{CL}$. We also show equivalent results for *non-deterministic* and *randomized catalytic space*.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Complexity classes

Keywords and phrases Space complexity, catalytic computation, error-correcting codes

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.50

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/138/`

Funding *Marten Folkertsma*: Supported by the Dutch Ministry of Economic Affairs and Climate Policy (EZK), as part of the Quantum Delta NL program.
Ian Mertz: Supported by Royal Society University Research Fellowship URF\R1\191059.
Florian Speelman: Supported by the Dutch Ministry of Economic Affairs and Climate Policy (EZK), as part of the Quantum Delta NL program, and the project Divide and Quantum "D&Q" NWA.1389.20.241 of the program "NWA-ORC", which is partly funded by the Dutch Research Council (NWO).
Quinten Tupker: Supported by the Dutch National Growth Fund (NGF), as part of the Quantum Delta NL program.

Acknowledgements We acknowledge useful conversations with Swastik Kopparty and Geoffrey Mon about error correction codes and how to apply them.

1 Introduction

1.1 Catalytic computation

Within space-bounded computation, the *catalytic computing* framework, first introduced by Buhrman, Cleve, Koucký, Loff, and Speelman [4], models the question of whether or not full memory can be a computational resource. Their main object of study is a *catalytic logspace* (CL) machine, in which a traditional logspace-bounded Turing machine is given access to a second work tape, polynomial in length, called the catalytic tape; while this tape is exponentially longer than the logspace work tape, it is already full with some string τ at the outset, and this string τ must be preserved by the overall computation.

Surprisingly, [4] show that CL can be much more powerful than L, with the catalytic tape being at least as powerful a resource as non-determinism ($\mathsf{NL} \subseteq \mathsf{CL}$), randomness ($\mathsf{BPL} \subseteq \mathsf{CL}$), and more ($\mathsf{TC}^1 \subseteq \mathsf{CL}$). They also showed that its power is nevertheless limited and falls far short PSPACE, namely $\mathsf{CL} \subseteq \mathsf{ZPP}$.

This work spawned a long sequence of explorations of the power of catalytic space. Given the base model of CL there are many possible variations and structural questions to be asked, such as the power of randomness [11, 6], non-determinism [5], non-uniformity [21, 23, 9, 10], and other variants [15, 2]. There have also been many works connecting the catalytic framework to broader questions in complexity theory, such as space-bounded derandomization [22, 14, 19], as well as adaptations of catalytic techniques to solve longstanding open questions such as compositional upper bounds for space [7, 8, 10] (see [18, 20] for surveys on the topic).

1.2 Lossy catalytic computation

Besides these more standard structural questions, there are also catalytic variants which are more specific to the catalytic space restriction. In particular, Gupta et al. [16] initiated the study of *lossy* catalytic computing, wherein the catalytic tape need not be exactly reset to its initial configuration. This model, which we refer to as LCSPACE, essentially asks how robust the core definition of catalytic space is to seemingly small relaxations; for example, in the *quantum* setting some computation error (albeit of a different form) is necessary for converting between different definitions based on allowed operations.

To begin, note that CL with $e \leq \text{poly}(n)$ errors trivially contains the class $\mathsf{SPACE}[e]$ by simply erasing the first e bits of the catalytic tape and using them as free memory. Because we have not managed to prove that any space-bounded class beyond L which is contained in ZPP, we should not expect to be able to prove CL is the same as CL with $e = \omega(\log n)$ errors. The question, then, is to understand where, in the range of $e = 0$ to $e = O(\log n)$, is the acceptable number of errors that CL can provably tolerate.

As an initial answer to the previous question, [16] show that CL gains no additional power from allowing any constant number of errors on the catalytic tape, i.e., $\mathsf{LCL}[O(1)] = \mathsf{CL}$. This remains the frontier of our knowledge, and Mertz [20] posed it as an open question to improve this result to any superconstant number of errors, or, alternatively, to provide evidence against being able to prove such a collapse.[1] Recently, Cook et al. [6] showed that a different error-prone model, namely *randomized* CL, is no more powerful than the base CL model.

[1] We cannot expect an unconditional separation between CL and any LCL, as even separating PSPACE from e.g. $\mathsf{TC}^1 (\subseteq \mathsf{CL})$ remains wide open.

1.3 Our results

In this work we completely characterize lossy catalytic space in terms of ordinary catalytic space. Let $\mathsf{CSPACE}[s, c]$ denote catalytic machines with free space s and catalytic space c, and let $\mathsf{LCSPACE}[s, c, e]$ be the same with up to e errors allowed in resetting the catalytic tape. We show that these e errors are equivalent to an additional $e \log c$ free bits of memory, up to constant factor losses.

▶ **Theorem 1.** *Let $s := s(n), c := c(n), e := e(n)$ be such that $e \leq c^{1-\Omega(1)}$. Then*

$$\mathsf{LCSPACE}[\Theta(s), \Theta(c), e] = \mathsf{CSPACE}[\Theta(s + e \log c), \Theta(c)]$$

Besides characterizing $\mathsf{LCSPACE}[s, c, e]$, the main takeaway of Theorem 1 is that allowing seemingly minor (superconstant) errors in the resetting condition can give an $\mathsf{LCSPACE}$ machine surprising power. A concrete instantiation of this view is the following direct corollary.

▶ **Corollary 2.** *For any $e := e(n)$,*

$$\mathsf{LCL}[e] = \mathsf{CL} \quad \textit{implies} \quad \mathsf{SPACE}[O(e \log n)] \subseteq \mathsf{ZPP}$$

If we revisit the assumption that we cannot hope to prove $\mathsf{SPACE}[e \log n]$ is in ZPP for any $e = \omega(1)$, then Corollary 2 implies the result of [16] is optimal with respect to e; any result of the form $\mathsf{LCL}[\omega(1)] = \mathsf{CL}$ is out of reach.

We also show that our proof extends to catalytic machines with additional power beyond errors, namely *non-deterministic* and *randomized* catalytic space.

▶ **Theorem 3.** *Let $\mathcal{C} \in \{\mathsf{NCSPACE}, \mathsf{BPCSPACE}\}$, and let $s := s(n), c := c(n), e := e(n)$ be such that $e \leq c^{1-\Omega(1)}$. Then*

$$\mathsf{LC}[s, c, e] = \mathcal{C}[\Theta(s + e \log c), \Theta(c)]$$

We briefly remark that the $e \leq c^{1-\Omega(1)}$ restriction in all our results is only needed to get the constant stretch in the catalytic tape; we discuss the unrestricted setting in Section 3.2.

1.4 Open problems

1.4.1 Errors in expectation

A related question asked in [20] is whether or not CL is equivalent to CL with $O(1)$ errors allowed *in expectation* over all starting catalytic tapes. This represents a different notion of distance between catalytic tapes, in opposition to Hamming distance, that may be more applicable to settings such as quantum computation. This question has received some attention in a related form by Bisoyi et al. [1], who introduce *almost* catalytic machines, which perfectly reset some catalytic tapes and are completely unrestricted on others.

However, no general results are known for expected errors – the results in [1] are very structured – and all techniques in our paper fail to restore the tape in pathological cases where a few starting tapes end up with potentially many errors. Furthermore, a barrier result was pointed out by an anonymous reviewer.[2]

[2] The main idea is that allowing virtually unlimited error in an exponentially small fraction of catalytic tapes gives us a strong form of the "compress-or-random" framework of previous papers; we can simulate

1.4.2 Randomized error-prone catalytic space

Recent work of Cook et al. [6] shows that $\mathsf{CSPACE}[s, c] = \mathsf{BPCSPACE}[O(s), \mathrm{poly}(c)]$, which, in conjunction with Theorem 3, seems to indicate that our theorems can be unified to show the connection between ordinary CSPACE and CSPACE which is both randomized and lossy, i.e. $\mathsf{CSPACE}[s + e \log c, c] = \mathsf{LBPCSPACE}[O(s), \mathrm{poly}(c), e]$. This would characterize how deterministic catalytic space handles both natural kinds of "error", namely both error in the output from the randomness and error in resetting the catalytic tape.

However, the proof of [6] only works when $c = 2^{\Theta(s)}$, and our connection to error-prone space incurs an $e \log c$ blowup in free space, putting us outside this regime. A generalization of their result, i.e. showing $\mathsf{CSPACE}[s, c] = \mathsf{BPCSPACE}[O(s), \mathrm{poly}(c)]$ for *every* s and c, would tie off this connection. Note that the polynomial blowup allowed in the catalytic tape means this result would not yield novel derandomization for ordinary space; even for $s, c = O(\log n)$ this would only show that derandomization overheads can be pushed into a polylogarithmic length catalytic tape, which was already shown by Pyne [22].

1.4.3 Lossy catalytic branching programs

Due to the flexibility in the conditions of Theorem 1, the results of Theorem 3 are likely to extend to other settings catalytic settings; for example, it is immediate to extend both results to CSPACE with *advice*. We focus on non-determinism and randomness simply because these are two of the most well-studied catalytic variants, and future works are free to adapt these proofs to their own settings.

In terms of notable omissions, however, one setting where one direction does not yet extend, and which is very related to advice, is the *catalytic branching program* model, which is a syntactic, and by extension non-uniform, way of capturing CSPACE. The issue here is simply that such machines can read and write their entire work tape in one step, which our simulation of CSPACE by $\mathsf{LCSPACE}$ is unequipped to handle. As we discuss in the full version of this paper, showing such branching programs are *reversible* would be sufficient to close this off.

1.4.4 Exact simulation space requirements

In the current simulation of errors using clean space, we use $4e \log c$ clean space. By contrast, in our simulation of clean space using errors, we use only e more errors. If errors can be simulated in clean space $e \log c$ instead, then there is only very low overhead in switching between the two perspectives. This would tighten the correspondence between errors and space that we establish. However, since the distance between two codewords required to correct e errors is $2e + 1$, a different error correction code would be necessary to reach clean space $e \log c$.

---

a randomized algorithm using the catalytic tape as our source of randomness, and in the exponentially unlikely event the tape is not sufficiently entropic we simply erase it and run brute force. Formalizing this intuition and combining it with the results of [6] gives a derandomization barrier to showing even $\mathsf{LCL}[O(1)] = \mathsf{CL}$, namely that randomized TC^1, which is not known to even be in P, would reduce to the lossy code problem.

2 Preliminaries

We begin by defining catalytic machines as introduced by Buhrman et al. [4].

▶ **Definition 4** (Catalytic space). *A catalytic Turing Machine is a space-bounded Turing machine with two work tapes: 1) a read-write work tape of length $s(n)$ which is initialized to $0^{s(n)}$, and 2) a read-write catalytic tape of length $c(n) \leq 2^{s(n)}$ which is initialized to an arbitrary state $\tau \in \{0,1\}^{c(n)}$. On any input $x \in \{0,1\}^n$ and initial catalytic state τ, a catalytic Turing machine has the property that at the end of the computation on input x, the catalytic tape will be in the initial state τ.*

In this work we focus on a relaxation of catalytic space by Gupta, Jain, and Sharma [16], where we are allowed to make some errors in resetting the catalytic tape.

▶ **Definition 5** (Lossy catalytic space). *A lossy catalytic Turing Machine with $e(n)$ errors is a catalytic machine where at the end of the computation on any input $x \in \{0,1\}^n$ and initial catalytic state τ, instead of requiring that the catalytic tape be in state τ, the catalytic tape can be in any state τ' such that τ and τ' differ in at most $e(n)$ locations.*

Lastly we specify the basic complexity classes arising from our two catalytic definitions, as well as their specification to the "logspace" setting, where most research interest at the moment lies.

▶ **Definition 6.** *We write*

- $\mathsf{CSPACE}[s, c]$*: the class of languages which can be recognized by catalytic Turing Machines with work space $s := s(n)$ and catalytic space $c := c(n)$.*
- $\mathsf{LCSPACE}[s, c, e]$*: the class of languages which can be recognized by lossy catalytic Turing Machines with work space $s := s(n)$, catalytic space $c := c(n)$, and $e := e(n)$ errors.*

We additionally write

- $\mathsf{CL} := \mathsf{CSPACE}[O(\log n), \operatorname{poly} n]$
- $\mathsf{LCL}[e] := \mathsf{LCSPACE}[O(\log n), \operatorname{poly} n, e]$

We note that throughout this paper we write $\mathcal{C}[O(f(n))]$ as a shorthand for $\bigcup_{c\in\mathbb{N}} \mathcal{C}[c \cdot f(n)]$ for complexity class $\mathcal{C}$ and function $f(n)$, and similarly for classes with multiple parameters or other asymptotics (e.g. $\mathsf{CSPACE}[\Theta(s), \Theta(c)]$).

3 Main theorem

In this section we will prove Theorem 1. We will do so via a simulation argument for each direction in turn.

3.1 Simulating errors with space

First, we show that $\mathsf{LCSPACE}[s, c, e] \subseteq \mathsf{CSPACE}[O(s + e \log c), O(c)]$. In fact, we will not need any increase in the length of our catalytic tape.

▶ **Theorem 7.** *Let $s := s(n), c := c(n), e := e(n)$. Then*

$$\mathsf{LCSPACE}[s, c, e] \subseteq \mathsf{CSPACE}[s + O(e \log c), c]$$

We note that this was also proven in [16] for the case of $\mathsf{LCL}[O(1)]$, but we will pursue a different proof, based on error-correcting codes, which will allow us to generalize to other catalytic models in Section 4.

Proof. Let M_e be an $\mathsf{LCSPACE}[s, c, e]$ machine. We will devise a $\mathsf{CSPACE}[s + O(e \log c), c]$ machine M_0 which simulates M_e. Note that in this section, we will not use our parameter restriction on e; this direction holds for every setting of s, c, and e. We will presume that $e \leq \frac{c}{\log(c)}$, as the inclusion becomes trivial otherwise.

Our simulation will go via an error-correcting code. In particular we will use *BCH codes*[3] (BCH), named after Bose, Ray-Chaudhuri, and Hocquenghem [3, 17], which we define as per [13, 12]. We define the mapping BCH and prove the following lemma in the appendix to the the full version of our paper.

▶ **Lemma 8.** *Let $q := 2^{\lceil \log(c+e) \rceil}$. There exists a mapping $\mathsf{BCH} : \mathbb{F}_q^q \to \mathbb{F}_q^q$ with the following operations:*

- ***Encoding:*** $\mathsf{Enc}_{\mathsf{BCH}}$ *takes as input a string S of length c, plus an additional $(2e+1)\lceil \log(c+e) \rceil$ bits initialized in 0, and outputs a codeword S_{enc}:*

  $$S + [0]_{(2e+1)\lceil \log(c+e) \rceil} \to_{\mathsf{Enc}} S_{enc}$$

  Furthermore, all outputs S_{enc} generated this way have minimum distance $\delta := 2e+1$ from one another.
- ***Decoding:*** $\mathsf{Dec}_{\mathsf{BCH}}$ *takes as input a string S'_{enc} of length $c + (2e+1)\log(c+e)$, with the promise that there exists a string S of length c such that $\mathsf{Enc}_{\mathsf{BCH}}(S + [0]_{2e \log(c+e)})$ differs from S'_{enc} in at most $\delta/2 - 1 = e$ locations, and outputs this string S:*

  $$S'_{enc} \to_{\mathsf{Dec}} S + [0]_{(2e+1)\log(c+e)}$$

Furthermore, both $\mathsf{Enc}_{\mathsf{BCH}}$ and $\mathsf{Dec}_{\mathsf{BCH}}$ are in place replacements of the input strings, they require at most an additional $O(e \log c)$ free space of memory.

We now move on to the simulation of our $\mathsf{LCSPACE}[s, c, e]$ machine M_e. Our $\mathsf{CSPACE}[s + O(e \log c), c]$ machine M_0 acts as follows:

1. **Initialization:** use the function $\mathsf{Enc}_{\mathsf{BCH}}$ to encode the initial state τ of the catalytic tape into a codeword, using $(2e+1)\lceil \log(c+e) \rceil$ additional bits from clean space,

   $$\tau + [0]_{(2e+1)\lceil \log(c+e) \rceil} \to_{\mathsf{Enc}} \tau_{enc}.$$

2. **Simulation:** Run M_e using clean space s and the first c bits of τ_{enc} as the catalytic tape. When M_e finishes the calculation, we record the answer in a bit of the free work tape. The catalytic tape is, at this point, in a state τ'_{enc} which differs in at most e locations from τ_{enc}.
3. **Cleanup:** use the function $\mathsf{Dec}_{\mathsf{BCH}}$ to detect and correct our resulting catalytic tape τ'_{enc}:

   $$\tau'_{enc} \to_{\mathsf{Dec}} \tau + [0]_{(2e+1)\lceil \log(c+e) \rceil}$$

   Once we finish this process, we output our saved answer and halt.

The correctness of M_0 is clear, as it gives the same output as M_e. By our error guarantee on M_e and the correctness of Dec, our catalytic tape is successfully reset to τ. Our catalytic memory is c as before, while for our free work space we require s bits to simulate M_e, an additional $(2e+1)\lceil \log(c+e) \rceil = (2 + o(1))e \log c$ zero bits for our codewords, and $O(e \log c)$ space for $\mathsf{Enc}_{\mathsf{BCH}}$ and $\mathsf{Dec}_{\mathsf{BCH}}$, for $s + O(e \log c)$ space in total. ◀

[3] Technically because of our parameters, they can even be considered Reed-Solomon codes, which are a special case of BCH codes; nevertheless we follow the presentation of the more general code form.

▶ Note 9. There is an alternative proof of this point, one which gets better parameters and relies on an interesting characterization of space, namely the *reversibility* of space. This proof is a simplification and extension of the one originally provided in [16], and we provide it in the appendix to the full version of our paper for those interested.

3.2 Simulating space with errors

We now show the other direction of Theorem 1, i.e. $\mathsf{CSPACE}[s + e\log c, c]$ is contained in $\mathsf{LCSPACE}[O(s), O(c), O(e)]$.

▶ **Theorem 10.** *Let $s := s(n), c := c(n), e := e(n)$, and $\epsilon > 0$ be such that $e = o(c^{\epsilon/(1+\epsilon)})$. Then*

$$\mathsf{CSPACE}[s + e\log c, c] \subseteq \mathsf{LCSPACE}[s + \log c, (1 + o(1))c, (1 + \epsilon)e]$$

Since $s \geq \log c$ by the definition of a catalytic machine, this achieves the reverse direction of Theorem 1 with very small blowups in s and c, and for e bounded by a small polynomial in c we get a negligible error blowup as well. Note that we allow $\epsilon > 1$, and so our proof is not limited to $e < c^{1/2}$; however, we will pay for larger values of e in the error blowup, and for $e = c^{1-o(1)}$ this factor becomes superconstant.

To understand our construction, we will first prove a version with looser space parameters. This result is incomparable to Theorem 10; although we lose a factor of e in our catalytic space, in exchange we have no restrictions on e and no loss in e either.

▶ **Theorem 11.** *Let $s := s(n), c := c(n), e := e(n)$ be such that c is a power of 2. Then*

$$\mathsf{CSPACE}[s + e\log c, c] \subseteq \mathsf{LCSPACE}[s + (\log e + \log c + 2), (e + 1)c, e]$$

Proof. Let M_0 be a $\mathsf{CSPACE}[s + e\log c, c]$ machine. We will devise a $\mathsf{LCSPACE}[s + (\log e + \log c + 2), (e+1)c, e]$ machine M_e which simulates M_0. By the definition of a Turing machine, we will assume that in any time step M_0 only reads and writes at most one bit on the work tape.

Throughout this proof, we will associate $[2^k]$ with $\{0,1\}^k$ in the usual manner, i.e. subtracting 1 and taking the binary representation, and so we will use them interchangeably. Our workhorse is the following folklore[4] construction:

▶ **Lemma 12.** *For every k, there exists a mapping $\mathsf{mem} : \{0,1\}^{2^k} \to \{0,1\}^k$, computable in space $k + 1$, such that the following holds: for any $\tau \in \{0,1\}^{2^k}$ and any $y \in \{0,1\}^k$,*

$$\mathsf{mem}(\tau \oplus \sigma_y) = \mathsf{mem}(\tau) \oplus y$$

where σ_y is the vector of length 2^k with a single 1 in position y.

Intuitively, Lemma 12 gives us an easily computable mapping where any transformation of the k-bit output string can be achieved by flipping one bit of the 2^k-bit input string, with the location of this single bitflip being determined only by the locations where the current and target output strings differ.[5]

[4] This construction is based on the solution to the so-called "almost impossible chessboard puzzle"; interested readers can find the setup and solution in videos on the YouTube channels 3Blue1Brown (`https://www.youtube.com/watch?v=wTJI_WuZSwE`) and Stand-up Maths (`https://www.youtube.com/watch?v=as7Gkm7Y7h4`). It can also be seen as the syndrome of the Hamming code.

[5] It is not particularly important that the location to flip in τ to achieve $\mathsf{mem}(\tau) \oplus y$ is exactly σ_y, but only that there exists *some* such place to flip, depending only on the mask y, and that this location can be easily calculated given y.

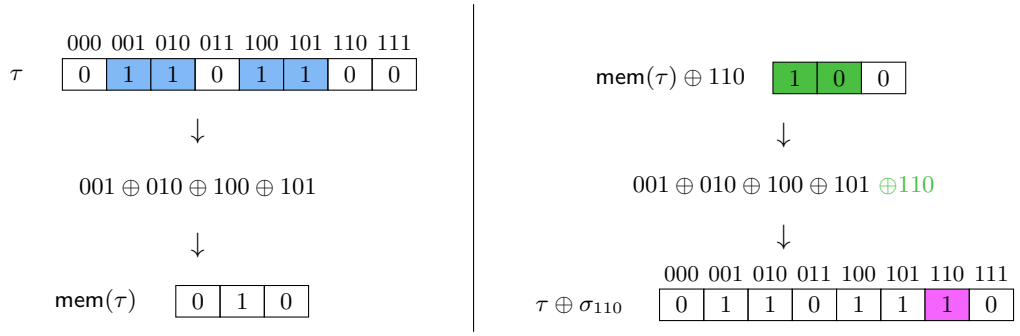

Figure 1 Example of our construction in Lemma 12 for $k = 3$ and $\tau = 01101100$: 1) calculating $\mathsf{mem}(\tau)$ based on the positions of the 1s in τ (blue); 2) how flipping one bit of τ (magenta) allows us to change $\mathsf{mem}(\tau)$ (changes in green).

Proof of Lemma 12. Let $\tau \in \{0,1\}^{2^k}$ be indexed by bitstrings $z \in \{0,1\}^k$. We will define our mapping mem as the entrywise sum of all indices z where $\tau_z = 1$, i.e.

$$\mathsf{mem}(\tau)_j = \bigoplus_{\substack{z \in \{0,1\}^k \\ z_j = 1}} \tau_z$$

This is clearly computable in space $k+1$, as we need only store z and our current sum. Now note that for any y, flipping the entry τ_y, i.e. $\tau \oplus \sigma_y$, flips every $\mathsf{mem}(\tau)_j$ entry where $y_j = 1$ and leaves all other $\mathsf{mem}(\tau)_j$ entry unchanged, which gives $\mathsf{mem}(\tau) \oplus y$ as claimed. ◀

At a high level, our $\mathsf{LCSPACE}[s + (\log e + \log c + 2), (e+1)c, e]$ machine M_e works as follows. M_e will use its s bits of free memory and the first c bits of catalytic memory to represent their equivalent blocks in M_0, i.e. s bits of free memory and c bits of catalytic memory. We will break the remaining $e \cdot c$ bits of our catalytic tape of M_e into e blocks $B_1 \dots B_e$ of size c each, and we denote by τ_i the memory in block B_i. We apply Lemma 12 with $k = \log c$ – recall that we assume c is a power of 2 – and use each $\mathsf{mem}(\tau_i)$ to represent an additional $\log c$ bits of free memory, giving us an additional $e \log c$ bits of memory in total. Our additional workplace memory will be used to compute the mapping, serve as pointers, and other assorted tasks.

Before formally stating M_e, we mention a special case of the construction of Lemma 12, which will allow us to use it in a reversible operational manner.

▷ Claim 13. Let $\tau_0, \tau_1, \tau_2 \dots \tau_{t-1}, \tau_t \in \{0,1\}^{2^k}$ be such that 1) τ_i and τ_{i-1} differ in exactly one coordinate for all $i \in [t]$; 2) $\mathsf{mem}(\tau_i)$ and $\mathsf{mem}(\tau_{i-1})$ differ in exactly one coordinate for all $i \in [t]$; and 3) $\mathsf{mem}(\tau_t) = \mathsf{mem}(\tau_0)$. Then $\tau_t = \tau_0$.

Proof. For all $i \in [t]$, let $b_i \in [k]$ be the location where $\mathsf{mem}(\tau_i)$ and $\mathsf{mem}(\tau_{i-1})$ differ, and let β_i be the location where τ_i and τ_{i-1} differ. Since $\mathsf{mem}(\tau_t) = \mathsf{mem}(\tau_0)$, each value $j \in [k]$ must appear an even number of times in the list $b_1 \dots b_t$, and since flipping any location $j \in [k]$ in $\mathsf{mem}(\tau)$ can only be obtained from one flip in τ at the unique location $2^j \in [2^k]$, it follows that each value $J \in [2^k]$ must appear an even number of times in the list $\beta_1 \dots \beta_t$. This means that τ_t is τ_0 with each bit flipped an even number of times, or in other words $\tau_t = \tau_0$. ◁

We now concretely define our machine M_e:

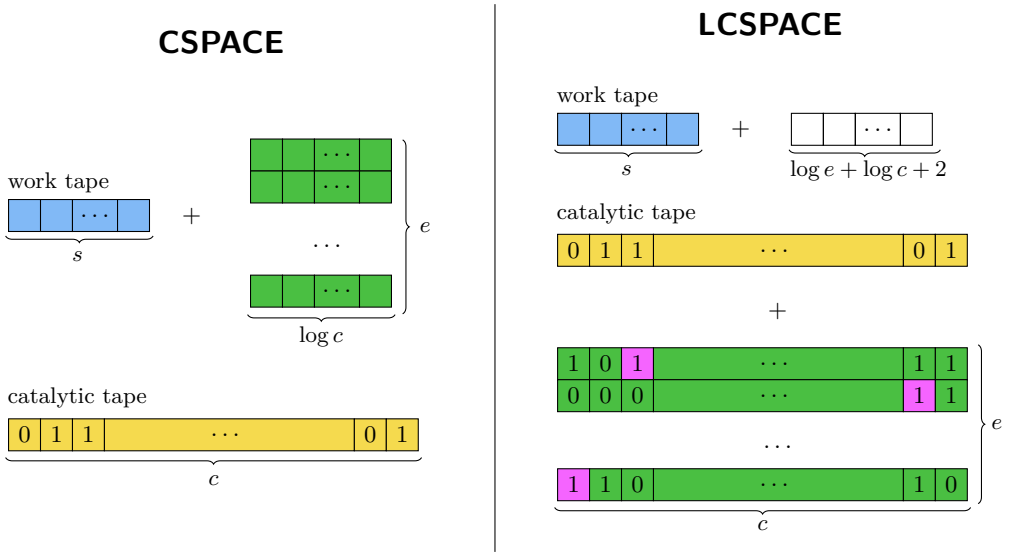

Figure 2 Structure of simulation, where like colors mean space used for same purpose. s bits of work space (blue) and c bits of catalytic space (gold) of the CSPACE machine are directly simulated in the LCSPACE machine, while each $\log c$ length chunk of additional work space of the CSPACE machine (green) is simulated by c catalytic bits and one error (magenta) of the LCSPACE machine. An additional $\log e + \log c + 2$ bits of work memory are used by the LCSPACE machine for scratch work, such as computing mem and addressing into the catalytic memory.

1. **Initialization:** for each block B_i, calculate mem_i and flip the mem_ith element of B_i:

$$\tau_i \to \tau_i \oplus \sigma_{\mathsf{mem}(\tau_i)} \qquad \forall i \in [e]$$

   Define τ_i^{enc} to be the memory after this process. Note that we now have exactly e errors on the tape, one in each τ_i^{enc}, and we are guaranteed that

$$\mathsf{mem}(\tau_i^{enc}) = 0^{\log c} \qquad \forall i \in [e]$$

2. **Simulation:** run M_0 using s free work bits and c catalytic bits, with the concatenation of the values mem_i as the other $e \log c$ free work bits. To do this, whenever we read or write a bit in our $e \log c$ bits of memory, we find the B_i responsible for this bit, calculate mem_i, and update τ_i using one bitflip to reflect how mem_i changes according to the operation of M_0:

$$\tau_i^{enc} \to \tau_i^{enc} \oplus \sigma_{2^j} \qquad \text{update occurs in bit } j \text{ of block } i$$

   If mem_i is unchanged, we make no updates to τ_i.
3. **Cleanup:** when we reach the end of M_0's computation, record the answer on the free work tape and set each mem_i value to $0^{\log c}$ one bit at a time:

$$\tau_i^{enc} \to \tau_i^{enc} \oplus \sigma_{2^j} \qquad \forall i \in [e], j : \mathsf{mem}(\tau_i^{enc}))_j = 1$$

   Once we finish this process, we output our saved answer and halt.

The correctness of M_e is clear, as we output the same value as M_0. Our catalytic space usage is $c + ec$ by construction, while our free space usage is s to simulate M_0, one extra bit to save the output, and any additional space required to handle the simulation of the

additional $e \log c$ work bits. In particular, we need $\log e$ bits for a pointer into B_i and $\log c + 1$ bits for the computation of mem_i by Lemma 12, for a total space usage of $s + \log e + \log c + 2$ as claimed.

We also claim that our lossy catalytic condition is satisfied. By the property of M_0, we make no errors on the c catalytic bits used for the simulation of M_0's catalytic space. The initialization step introduces at most one error per τ_i, thus giving at most e errors on to the catalytic tape. After the initialization step, each other update to τ_i corresponds to changing a single bit in $\mathsf{mem}(\tau_i)$, with the final value being the same as the value after initialization. Thus by Claim 13 we restore the catalytic tape to its position after the initialization phase, and so our end state corresponds to our original catalytic tape with at most e errors, i.e. those induced by the initialization phase, as required. ◀

We now return to Theorem 10, which requires only a small modification of the above proof, namely to break the the catalytic tape into more, smaller blocks, which reduces its required length at the cost of a few extra errors. This modification works because the number of pure bits represented is logarithmic in the length of the block, and so making the blocks smaller barely affects the number of bits represented; for example, $c/2$ bits in a block still lets you represent $\log(c) - 1$ bits, so half the size only loses one bit per block.

Proof of Theorem 10. Let M_0 be a $\mathsf{CSPACE}[s + e \log c, c]$ machine. We will devise a $\mathsf{LCSPACE}[s + e \log c, (1 + o(1))c, (1 + \epsilon)e]$ machine M_e which simulates M_0, where ϵ satisfies $e = o(c^{\epsilon/(1+\epsilon)})$.

An easy manipulation gives us $c = \omega(e^{1+1/\epsilon})$, and so there exists a function $\delta = \omega(1)$ such that $c \geq (\delta e)^{1+1/\epsilon}$. We will have the same approach as Theorem 11, but now we use $(1 + \epsilon)e$ blocks of length $2^{\lceil \log c/(\delta e) \rceil}$ each, i.e., using Lemma 12 for $k = \lceil \log c/(\delta e) \rceil$. Since we introduce one error per block, the number of errors the machine makes is at most $(1 + \epsilon)e$, while our total catalytic tape has length

$$c + (1 + \epsilon)e \cdot 2^{\lceil \log c/(\delta e) \rceil} \leq c + (1 + \epsilon)e \cdot \frac{2c}{\delta e} = c \cdot (1 + o(1))$$

Lastly, we can use our additional catalytic memory to simulate a work tape of length

$$(1 + \epsilon)e \cdot \log 2^{\lceil \log(c/\delta e) \rceil} \geq e \cdot (1 + \epsilon) \log \frac{c}{\delta e}$$

and by manipulating our starting assumption we get that

$$\begin{aligned} c &\geq (\delta e)^{1+1/\epsilon} \\ (c/\delta e)^{\epsilon} &\geq \delta e \\ \epsilon \log(c/\delta e) &\geq \log \delta e \\ \log c - \log \delta e + \epsilon \log(c/\delta e) &\geq \log c \\ e \cdot (1 + \epsilon) \log(c/\delta e) &\geq e \log c \end{aligned}$$

thus giving us a simulation of $e \log c$ work bits as claimed. The correctness and lossy catalytic condition can then be argued as above, and our free space usage is s plus an additional $\log(c \cdot o(1)) + 2 \leq \log c$ bits, for a total space usage of $s + \log c$ as claimed. ◀

4 Further consequences

With this, we have concluded our main theorem and proof. We now move to corollaries and extensions.

4.1 Lossy catalytic logspace with superconstant errors

As stated in the introduction, it immediately follows from Theorem 1 that proving $\mathsf{LCL}[e] = \mathsf{CL}$ is likely difficult, if not false, for superconstant values of e.

Proof of Corollary 2. This follows immediately from the fact that

$$\begin{aligned}\mathsf{LCSPACE}[O(\log n), \operatorname{poly} n, e] &= \mathsf{CSPACE}[O(\log n + e \log(\operatorname{poly} n)), \operatorname{poly} n] \\ &= \mathsf{CSPACE}[O(e \log n), \operatorname{poly} n] \\ &\supseteq \mathsf{SPACE}[O(e \log n)]\end{aligned}$$

combined with the fact that $\mathsf{CL} \subseteq \mathsf{ZPP}$ by [4]. ◀

4.2 Lossy catalytic space with other resources

As mentioned in Section 1, there are many extensions of the base catalytic model besides LCSPACE, such as randomized, non-deterministic, and non-uniform CSPACE. So far, however, there has been little discussion of classes where more than one such external resource has been utilized. In this section, we will discuss two of the aforementioned models – randomized and non-deterministic CSPACE – in the presence of errors.

▶ **Definition 14.** *Let f be a Boolean function on n inputs.*

- *A non-deterministic catalytic Turing machine for f is a catalytic machine M which, in addition to its usual input x, has access to a read-once witness string w, of length at most exponential in the total space of M, based on x, which has the following properties:*
  - *if $f(x) = 1$, then there exists a witness w such that $M(x, w) = 1$*
  - *if $f(x) = 0$, then for every witness string w, $M(x, w) = 0$*

  Furthermore, for every witness string w, $M(x, w)$ restores the catalytic tape to its original configuration.
- *A randomized catalytic Turing machine for f is a catalytic machine M which, in addition to its usual input x, has access to a read-once uniformly random string r, of length at most exponential in the total space of M, such that*

$$\mathbf{Pr}_r[M(x, r) = f(x)] \geq 2/3.$$

  Furthermore, for every witness string w, $M(x, w)$ restores the catalytic tape to its original configuration.

We write

- $\mathsf{NCSPACE}[s, c]$*: the class of languages which can be recognized by non-deterministic catalytic Turing Machines with work space $s := s(n)$ and catalytic space $c := c(n)$.*
- $\mathsf{BPCSPACE}[s, c]$*: the class of languages which can be recognized by randomized catalytic Turing Machines with work space $s := s(n)$ and catalytic space $c := c(n)$.*

Furthermore, for $\mathcal{C} \in \{\mathsf{NCSPACE}, \mathsf{BPCSPACE}\}$, we define $\mathsf{LC}[s, c, e]$ to be the class $\mathcal{C}[s, c]$ with the catalytic resetting definition replaced by the LCSPACE *resetting definition, i.e., where e errors are allowed to remain on the catalytic tape at the end of any computation.*

Note that we allow the errors to depend on the witness and randomness, respectively. Without delving too deep into these models, however, it is clear that our proof of Theorem 1 carries over to all the above definitions.

Proof sketch of Theorem 3. As earlier, we need to show both directions. We will prove the same two equivalences as in Theorems 7 and 10, namely

1. $\mathsf{L}\mathcal{C}[s, c, e] \subseteq \mathcal{C}[s + O(e \log c), c]$
2. $\mathcal{C}[s + e \log c, c] \subseteq \mathsf{L}\mathcal{C}[s + \log c, (1 + o(1))c, (1 + \epsilon)e]$

Both directions will follow immediately via the same simulation as before. For the forward direction, we simulate our $\mathsf{L}\mathcal{C}$ machine by a $\mathcal{C}$ machine as usual, and then at the last step we correct the changes using our BCH code as before; this works just as before because the code allow us to correct any e errors on the catalytic tape, regardless of how they came about.

For the reverse direction, again our $\mathcal{C}$ machine will only read or write one bit per time step, and so we use the same mem_i approach to simulating our additional $e \log c$ bits of free memory, which does not change based on the operation of the rest of the machine. As in the forward direction, simulating the actual workings of the $\mathcal{C}$ machine via the $\mathsf{L}\mathcal{C}$ machine, given our method of simulating the $e \log c$ additional bits of memory, is straightforward, and our resetting step at the end again resets our extra catalytic tape regardless of the computation path the $\mathcal{C}$ machine takes. ◀

References

1. Sagar Bisoyi, Krishnamoorthy Dinesh, Bhyabya Deep Rai, and Jayalal Sarma. Almost-catalytic and property-catalytic computation, 2024.
2. Sagar Bisoyi, Krishnamoorthy Dinesh, and Jayalal Sarma. On pure space vs catalytic space. *Theoretical Computer Science (TCS)*, 921:112–126, 2022. doi:10.1016/J.TCS.2022.04.005.
3. Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and control*, 3(1):68–79, 1960. doi:10.1016/S0019-9958(60)90287-4.
4. Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014. doi:10.1145/2591796.2591874.
5. Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space: Non-determinism and hierarchy. *Theory of Computing Systems (TOCS)*, 62(1):116–135, 2018. doi:10.1007/S00224-017-9784-7.
6. James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. *Electronic Colloquium on Computational Complexity (ECCC)*, TR24-106, 2024. URL: https://eccc.weizmann.ac.il/report/2024/106.
7. James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *ACM Symposium on Theory of Computing (STOC)*, pages 752–760. ACM, 2020. doi:10.1145/3357713.3384316.
8. James Cook and Ian Mertz. Encodings and the tree evaluation problem. *Electronic Colloquium on Computational Complexity (ECCC)*, TR21-054, 2021. URL: https://eccc.weizmann.ac.il/report/2021/054.
9. James Cook and Ian Mertz. Trading time and space in catalytic branching programs. In *IEEE Conference on Computational Complexity (CCC)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:21, 2022. doi:10.4230/LIPIcs.CCC.2022.8.
10. James Cook and Ian Mertz. Tree evaluation is in space O(log n · log log n). In *ACM Symposium on Theory of Computing (STOC)*, pages 1268–1278. ACM, 2024. doi:10.1145/3618260.3649664.
11. Samir Datta, Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Randomized and symmetric catalytic computation. In *CSR*, volume 12159 of *Lecture Notes in Computer Science (LNCS)*, pages 211–223. Springer, 2020. doi:10.1007/978-3-030-50026-9_15.

12 Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Syndrome encoding and decoding of bch codes in sublinear time, 2006. URL: https://www.cs.bu.edu/~reyzin/code/bch-excerpt.pdf.

13 Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology - EUROCRYPT 2004*, 2004.

14 Dean Doron, Edward Pyne, and Roei Tell. Opening up the distinguisher: A hardness to randomness approach for BPL = L that uses properties of BPL. In *Proceedings of the 56nd Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 2039–2049, 2024. doi:10.1145/3618260.3649772.

15 Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Unambiguous catalytic computation. In *Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.16.

16 Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Lossy catalytic computation. *Computing Research Repository (CoRR)*, abs/2408.14670, 2024. doi:10.48550/arXiv.2408.14670.

17 Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffers*, 2:147–156, 1959.

18 Michal Koucký. Catalytic computation. *Bulletin of the EATCS (B.EATCS)*, 118, 2016. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/400.

19 Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, to appear, 2024.

20 Ian Mertz. Reusing space: Techniques and open problems. *Bulletin of the EATCS (B.EATCS)*, 141:57–106, 2023. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/780.

21 Aaron Potechin. A note on amortized branching program complexity. In *IEEE Conference on Computational Complexity (CCC)*, volume 79 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CCC.2017.4.

22 Edward Pyne. Derandomizing logspace with a small shared hard drive. In *39th Computational Complexity Conference, CCC 2024*, volume 300 of *LIPIcs*, pages 4:1–4:20, 2024. doi:10.4230/LIPICS.CCC.2024.4.

23 Robert Robere and Jeroen Zuiddam. Amortized circuit complexity, formal complexity measures, and catalytic algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 759–769. IEEE, 2021. doi:10.1109/FOCS52979.2021.00079.